

# APLICACIÓN DE LAS ÁLGEBRAS DE CONFIGURACIÓN DE BRAUER AL TRANSMILENIO

Maria José Giraldo | Jose Felipe Sanabria

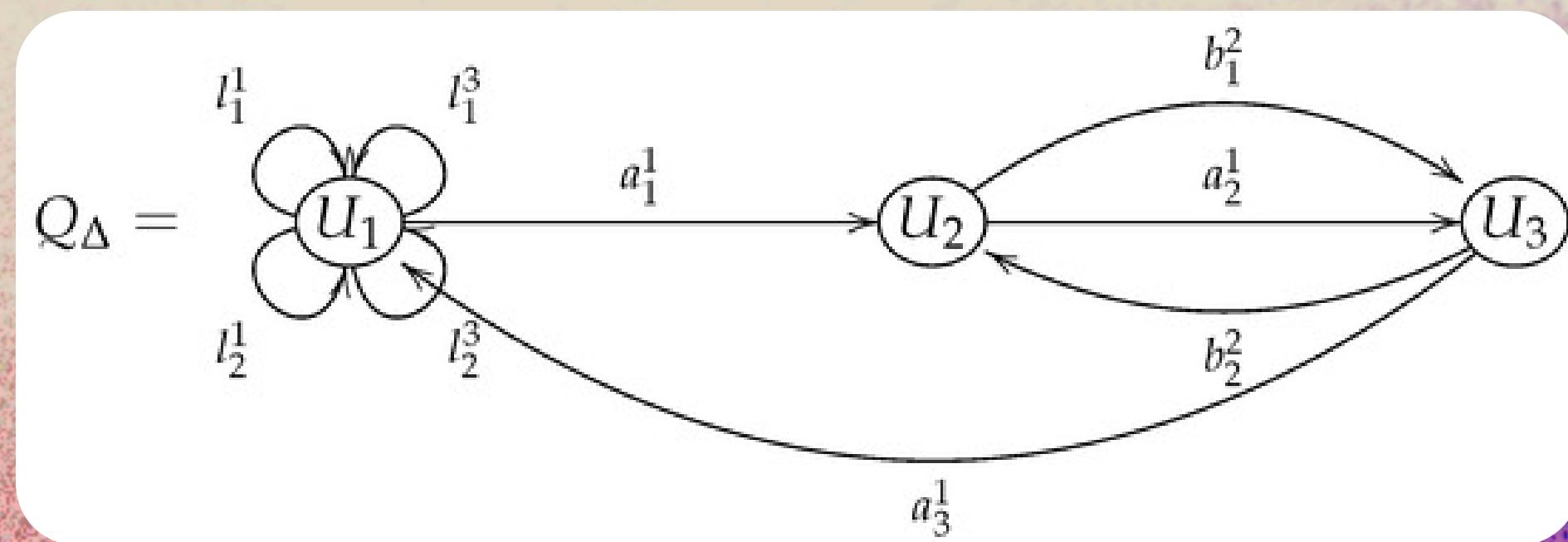
14 mayo 2025 | Computación científica I

# CONTENIDOS

1. Álgebras de configuración de Brauer: Construcción
2. Aplicaciones de las Álgebras de configuración
3. Transmilenio: Aplicación y base de datos
4. Resultados
5. Mejoras y desafíos

# ¿QUE ES UN ALGEBRA DE CONFIGURACION DE BRAUER?

Es un tipo de álgebra de representación de dimensión finita definida en 2017 por Green y Schroll, las cuales cumplen con una construcción y propiedades especiales.



# CONSTRUCCIÓN DE UNA ÁLGEBRA DE CONF.

## Anillo:

Sea  $A$  un conjunto con dos operaciones  $(+, *)$  entonces se dice que  $(A, +, *)$  es un anillo si y solo si:

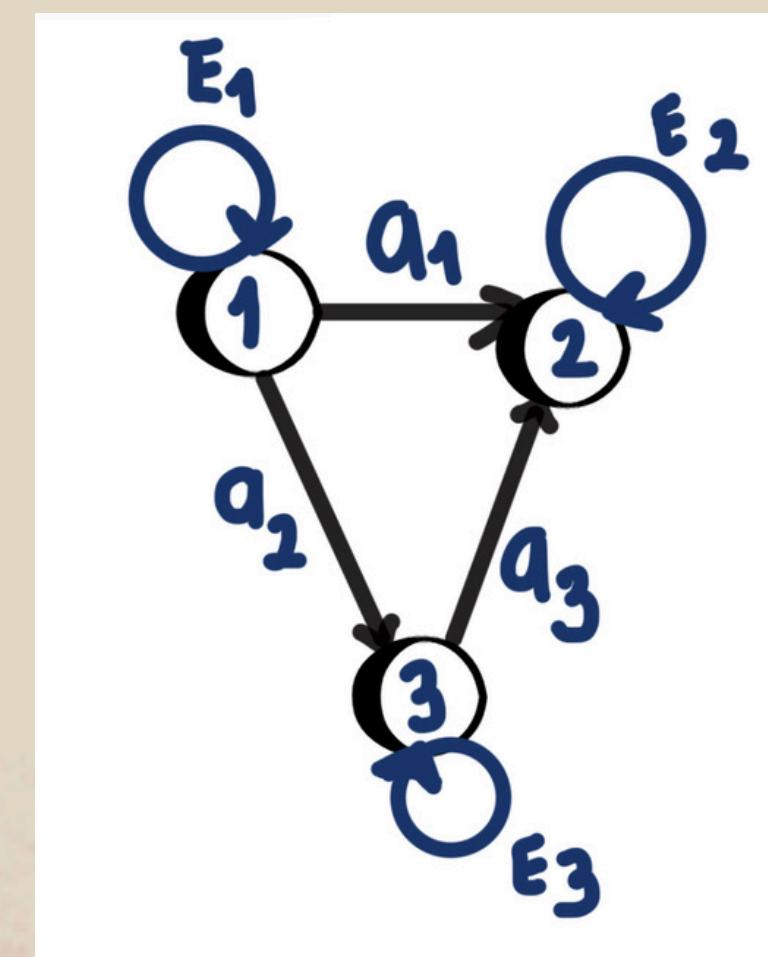
- $(A, +)$  es un grupo abeliano.
- $(A, *)$  es un monoide

## K-Algebra:

Dado un  $K$  campo.  $A$  es un anillo con identidad multiplicativa tal que  $A$  tiene estructura de  $K$ -espacio vectorial y es compatible con el producto del anillo.

## Grafo dirigido (Quiver):

Sea  $G := (V, E)$  un quiver donde  $V$  son vertices y  $E$  aristas.



## Flechas estacionarias:

Es una arista que genera un bucle en un nodo.

## Algebras de caminos:

Todos los caminos posibles a realizar entre todos los vértices. Estas álgebras generan un espacio :=  $\{E_1, E_2, E_3, a_1, a_2, a_3, a_2a_3\}$

# CONSTRUCCIÓN DE UNA ALGEBRA DE CONF.

## Álgebra de configuración de Brauer:

Se define una configuración de Brauer B como sigue:

$$B := \{B_0, B_1, \mu, \mathcal{O}\}$$

Donde:

- $B_0$  representa los vértices del grafo.
- $B_1$  representan polígonos con multiconjuntos de los vértices.
- $\mu$  es una función de  $B_0$  a los naturales definida:

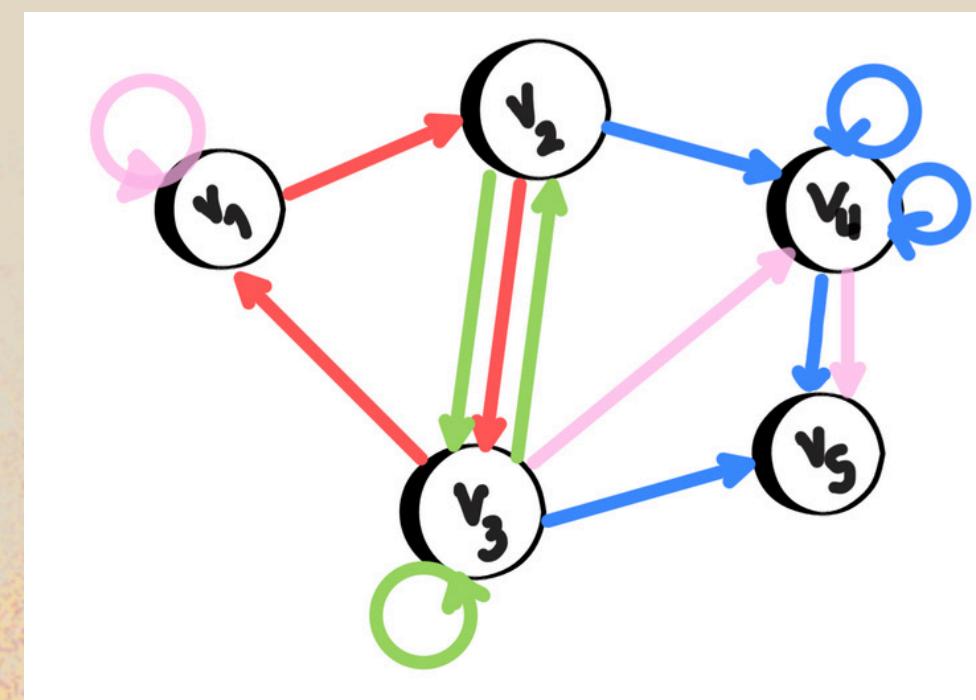
$$\begin{cases} 1 & \text{si } val(b_i) \neq 1 \\ 2 & \text{si } val(b_i) = 1 \end{cases}$$

Donde  $val(b_i)$  es la valencia.

## Algebra de Caminos de Brauer:

Es una KQ álgebra que limitamos con el Ideal donde no se pueden combinar caminos.

No se puede viajar por el mismo camino mas de una vez.



# CONSTRUCCIÓN DE UNA ÁLGEBRA DE CONF.

## Invariantes algebraicos:

Los invariantes son características matemáticas que se mantienen a través de transformaciones.

- **Dimensión del Álgebra**

$$2|B_1| + \sum_{\alpha \in B_0} val(\alpha) \left( \mu(\alpha)val(\alpha) - 1 \right)$$

- **Dimensión del centro:**

$$1 + |B_1| - |B_0| + \sum_{\alpha \in B_0} val(\alpha) + loops - |\mathcal{C}_n|$$

donde,

$$\mathcal{C}_n = \{\alpha \in B_0 : val(\alpha) = 1 \wedge \mu(\alpha) \neq 1\}$$

- **Entropía**

$$\sum_{\alpha \in B_0} \frac{val(\alpha)}{\sum_{\alpha \in B_0} \mu(\alpha)val(\alpha)} \log_2 \left( \frac{val(\alpha)}{\sum_{\alpha \in B_0} \mu(\alpha)val(\alpha)} \right)$$

# APLICACIONES

Se han usado para estudiar álgebra de tipo representación salvaje y para encontrar soluciones a generalizaciones del problema de los McNuggets, la teoría de representaciones, la criptografía y transporte.

# TRANSMILENIO: APLICACIÓN PRACTICA Y BASE DE DATOS

Usaremos las álgebras de configuración de Brauer para analizar y optimizar la ruta B12 en el tramo Norte de Transmilenio.

id_ruta	nombre_ruta	id_estacion	nombre_estacion	troncal	orden_parada	cia_anterior_m	corrido_prom_s	corrido_desv_s	espera_prom_s	espera_desv_s	semaforos	as_intermedias	edad_pasajeros	tienda_promedio	
1	B12	Portal Norte-Sur	TMB1202	Toberín	AN	2	11.046	3.660	301	198	30	0	0	301	230
2	B12	Portal Norte-Sur	TMB1203	Calle 100	AN	3	10.021	3.178	378	176	26	0	0	341	287
3	B12	Portal Norte-Sur	TMB1204	Calle 127	AN	4	9.085	3.037	280	112	17	0	0	198	158
4	B12	Portal Norte-Sur	TMB1205	Calle 142	AN	5	9.901	3.534	289	197	29	0	0	257	154
5	B12	Portal Norte-Sur	TMB1206	Calle 146	AN	6	6.464	2.704	267	203	30	0	0	222	222
6	B12	Portal Norte-Sur	TMB1207	Prado	AN	7	9.665	3.058	360	129	19	0	0	239	152
7	B12	Portal Norte-Sur	TMB1208	Alcalá	AN	8	10.239	3.473	311	218	33	0	0	157	157
8	B12	Portal Norte-Sur	TMB1209	Toberín 2	AN	9	11.162	3.719	356	207	31	0	0	283	153
9	B12	Portal Norte-Sur	TMB1210	Calle 100 2	AN	10	6.591	2.455	215	212	32	0	1	349	289
10	B12	Portal Norte-Sur	TMB1211	Calle 127 2	AN	11	8.343	2.913	278	213	32	0	1	303	287
11	B12	Portal Norte-Sur	TMB1212	Calle 142 2	AN	12	6.924	2.576	221	207	31	0	1	312	262
12	B12	Portal Norte-Sur	TMB1213	Calle 146 2	AN	13	9.775	3.871	321	165	25	0	1	284	171
13	B12	Portal Norte-Sur	TMB1214	Prado 2	AN	14	7.382	2.674	224	234	35	0	2	186	186

# TRANSMILENIO: APLICACIÓN PRACTICA Y BASE DE DATOS

Variables:

- id\_ruta
- nombre\_ruta
- id\_estacion
- nombre\_estacion\_troncal
- orden\_parada
- distancia\_anterior\_m
- tiempo\_recorrido\_prom\_s
- tiempo\_recorrido\_desv\_s
- tiempo\_espera\_prom\_s
- tiempo\_espera\_desv\_s
- semaforos\_paradas\_intermedias
- capacidad\_pasajeros
- demanda\_promedio



```
class ModifiedBrauerAlgebra:
    def __init__(self, stations_data):
        ...
        # B0: conjunto de vértices (estaciones)
        self.B0 = stations_data['nombre_estacion'].tolist()

        # Crear grafo dirigido
        self.graph = nx.MultiDiGraph() # Usamos MultiDiGraph para permitir múltiples bucles

        # Añadir nodos
        for station in self.B0:
            self.graph.add_node(station)

        # Añadir aristas basadas en el orden de las paradas
        for i in range(len(self.B0) - 1):
            source = self.B0[i]
            target = self.B0[i + 1]
            distance = stations_data.iloc[i+1]['distancia_anterior_m']
            time = stations_data.iloc[i+1]['tiempo_recorrido_prom_s']
            ...
            self.graph.add_edge(source, target, weight=distance, time=time)
```

```
...
for i, station in enumerate(self.B0):
    wait_time = stations_data.iloc[i]['tiempo_espera_prom_s']
    loops_count = max(1, int(round(wait_time / 60))) # Convertir segundos a minutos y redondear

    for j in range(loops_count):
        # Cada bucle representa un minuto de espera
        self.graph.add_edge(station, station, weight=0, time=60, type='wait')

# B1: representan polígonos con multiconjuntos de vértices B0
# En este caso, las aristas del grafo (incluyendo bucles)
# MultiDiGraph devuelve tuplas (u,v,key) así que necesitamos adaptarnos
self.B1 = list(self.graph.edges(data=True)) # Ahora incluye data

# Calcular valencias de los nodos
self.valences = dict()
for node in self.B0:
    in_degree = self.graph.in_degree(node)
    out_degree = self.graph.out_degree(node)
    self.valences[node] = in_degree + out_degree

# Función  $\mu$  según la definición dada
self.mu = dict()
for node in self.B0:
    if self.valences[node] != 1:
        self.mu[node] = 1
    else:
        self.mu[node] = 2

# Orientación 0 (ya definida por el grafo dirigido)
self.0 = self.B1

# Calcular invariantes algebraicos
self.compute_invariants()
```

```
...
# Dimensión del álgebra
    self.dimension_algebra = 0
    for node in self.B0:
        val = self.valences[node]
        mu_val = self.mu[node]
        self.dimension_algebra += val * (mu_val * val - 1)
    self.dimension_algebra += 2 * len(self.B1)

# Calcular bucles
loops = 0
for u, v, data in self.B1:
    if u == v:
        loops += 1

# Conjunto C_n
C_n = [node for node in self.B0 if self.valences[node] == 1 and self.mu[node] != 1]

# Dimensión del centro
self.dimension_center = 1 + len(self.B1) - len(self.B0)
for node in self.B0:
    self.dimension_center += self.valences[node]
self.dimension_center += loops - len(C_n)

# Entropía
total_mu_val = sum(self.mu[node] * self.valences[node] for node in self.B0)
if total_mu_val > 0:
    self.entropy = 0
    for node in self.B0:
        val = self.valences[node]
        p = val / total_mu_val
        if p > 0: # Evitar log(0)
            self.entropy -= p * math.log2(p)
else:
    self.entropy = 0
```



```
def analyze_route_metrics(brauer_algebra):

    stations_data = brauer_algebra.stations_data
    G = brauer_algebra.graph

    # Análisis de tiempos y distancias
    total_distance = stations_data['distancia_anterior_m'].sum()
    total_time = stations_data['tiempo_recorrido_prom_s'].sum()

    # Tiempo total de espera
    total_wait_time = stations_data['tiempo_espera_prom_s'].sum()

    # Conteo de bucles por estación
    station_loops = {}
    for node in brauer_algebra.B0:
        loops = [(u, v) for u, v, d in G.edges(data=True) if u == v and v == node]
        station_loops[node] = len(loops)

    # Calcular eficiencia basada en el álgebra de Brauer
    if brauer_algebra.dimension_algebra > 0:
        efficiency = brauer_algebra.entropy / brauer_algebra.dimension_algebra
    else:
```



```
def suggest_optimizations(brauer_algebra):

    stations_data = brauer_algebra.stations_data
    G = brauer_algebra.graph

    # Identificar estaciones con alto número de bucles y alta demanda
    critical_stations = []
    for station in brauer_algebra.B0:
        loops = [(u, v) for u, v, d in G.edges(data=True) if u == v and v == station]
        loops_count = len(loops)

        station_data = stations_data[stations_data['nombre_estacion'] == station]
        if not station_data.empty:
            wait_time = station_data['tiempo_espera_prom_s'].values[0]
            demand = station_data['demanda_promedio'].values[0]

            # Si tiene muchos bucles y alta demanda
            if loops_count > 0 and demand > stations_data['demanda_promedio'].mean():
                critical_stations.append((station, loops_count, wait_time, demand))

    print("\n== SUGERENCIAS DE OPTIMIZACIÓN (MODELO CÍCLICO) ==")

    if len(critical_stations) > 0:
        print("Estaciones críticas (alto tiempo de espera y demanda):")
        for station, loops_count, wait_time, demand in critical_stations:
            print(f" - {station}: Bucle = {loops_count}, Tiempo de espera = {wait_time:.2f}s, Demanda = {demand}")

    print("\nSugerencias:")
    for station, loops_count, wait_time, demand in critical_stations:
        print(f" - En la estación {station}:")
        print(f"   * Reducir tiempo de espera para disminuir el número de bucles")
        if demand > 200: # Umbral ajustado para alta demanda
            print(f"   * Aumentar la capacidad de servicio debido a la alta demanda ({demand} pasajeros)")

    else:
        print("No se identificaron estaciones críticas.")
```

```
# Calcular posibles reducciones de tiempo basadas en el modelo cíclico
cycle_optimization = {}

# Principal ciclo de la ruta (incluyendo retorno)
main_cycle = brauer_algebra.B0 + [brauer_algebra.B0[0]]
cycle_time = 0
cycle_distance = 0

# Calcular tiempo y distancia del ciclo principal
for i in range(len(main_cycle) - 1):
    source = main_cycle[i]
    target = main_cycle[i+1]

    # Buscar la arista correcta
    for u, v, data in G.edges(data=True):
        if u == source and v == target and data['type'] != 'wait':
            cycle_time += data['time']
            cycle_distance += data['weight']
            break

# Añadir tiempo de espera en estaciones
for station in brauer_algebra.B0:
    station_data = stations_data[stations_data['nombre_estacion'] == station]
    if not station_data.empty:
        cycle_time += station_data['tiempo_espera_prom_s'].values[0]

...
# Estimar posible ahorro
wait_time_savings = sum([wait_time * 0.2 for _, _, wait_time, _ in critical_stations])
print(f"\nPosible ahorro de tiempo con optimizaciones: {wait_time_savings:.2f}s ({wait_time_savings/60:.2f}min)")

# Visualizar estructura cíclica y bucles con todos los nodos
plt.figure(figsize=(14, 10))

# Usar layout circular para enfatizar la estructura cíclica
pos = nx.circular_layout(G)

# Escalado de nodos según demanda
node_sizes = []
node_colors = []
for node in G.nodes():
    station_data = stations_data[stations_data['nombre_estacion'] == node]
    demand = station_data['demanda_promedio'].values[0]
    # Tamaño basado en demanda (normalizado)
    node_sizes.append(max(300, 50 + demand * 1.2))

    # Color basado en si es estación crítica
    is_critical = any(station == node for station, _, _, _ in critical_stations)
    node_colors.append('orangered' if is_critical else 'lightblue')
```

# RESULTADOS

Total de aristas (B1): 28

- Aristas de ruta: 13
- Aristas de retorno: 1
- Bucles de espera: 14

Valencias de los nodos:

Portal Norte: 4

Toberín: 4

Calle 100: 4

Calle 127: 4

Calle 142: 4

Calle 146: 4

Prado: 4

Alcalá: 4

Toberín 2: 4

Calle 100 2: 4

Calle 127 2: 4

Calle 142 2:

Calle 146 2: 4

Prado 2: 4

Función  $\mu$ :

- $\mu(\text{Portal Norte}) = 1$
- $\mu(\text{Toberín}) = 1$
- $\mu(\text{Calle 100}) = 1$
- $\mu(\text{Calle 127}) = 1$
- $\mu(\text{Calle 142}) = 1$
- $\mu(\text{Calle 146}) = 1$
- $\mu(\text{Prado}) = 1$
- $\mu(\text{Alcalá}) = 1$
- $\mu(\text{Toberín 2}) = 1$
- $\mu(\text{Calle 100 2}) = 1$
- $\mu(\text{Calle 127 2}) = 1$
- $\mu(\text{Calle 142 2}) = 1$
- $\mu(\text{Calle 146 2}) = 1$
- $\mu(\text{Prado 2}) = 1$

Invariantes algebraicos:

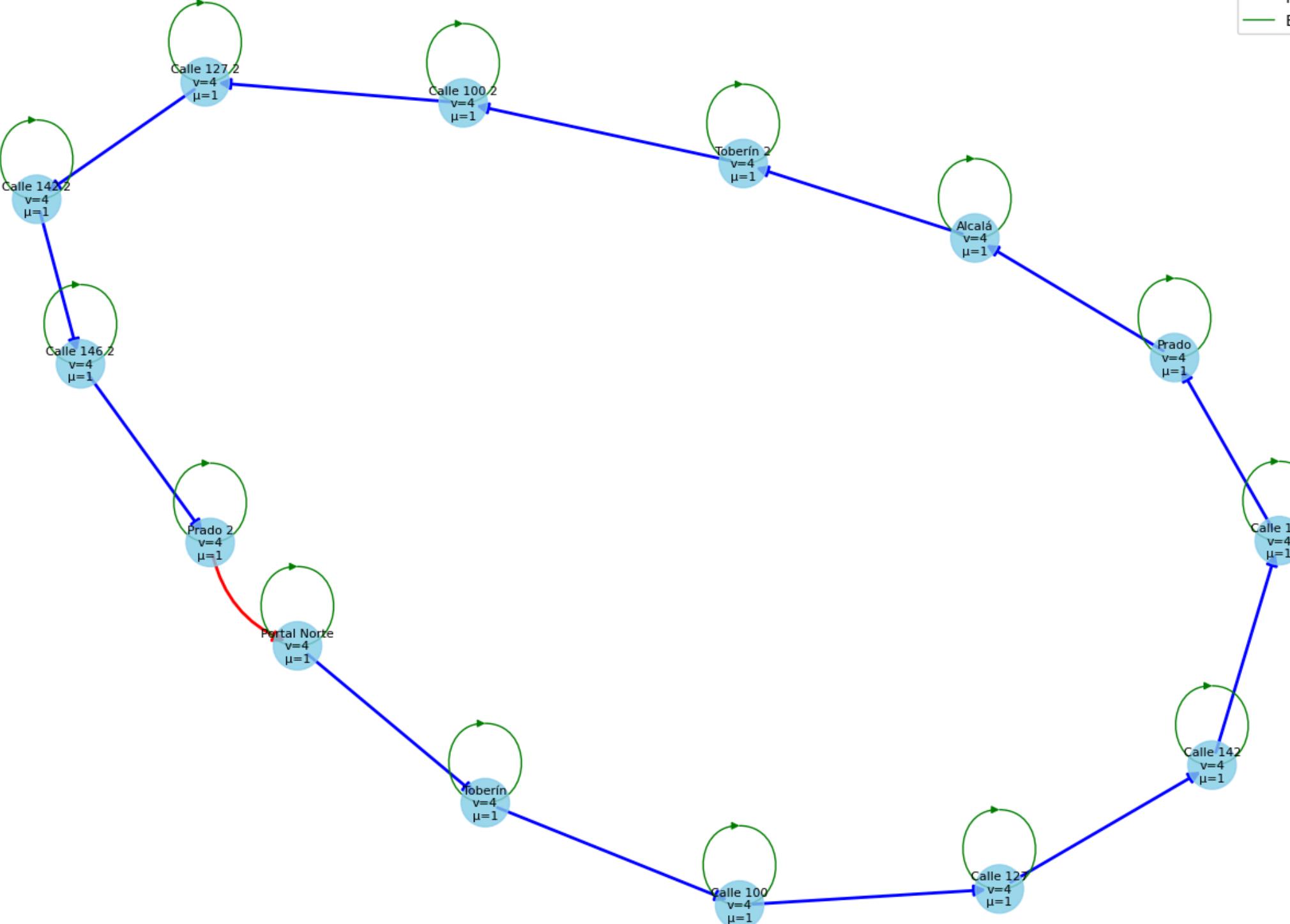
Dimensión del álgebra: 224

Dimensión del centro: 85

Entropía: 3.8074

Grafo de Brauer Modificado para la ruta B12 de Transmilenio

■ Ruta  
■ Retorno  
■ Espera (bucle)



# ANÁLISIS DE CICLOS

Número total de ciclos: 15

Ciclos de ruta: 1

Ciclos de espera: 14

# ANÁLISIS DE MÉTRICAS

Distancia total de la ruta: 11659.80 metros

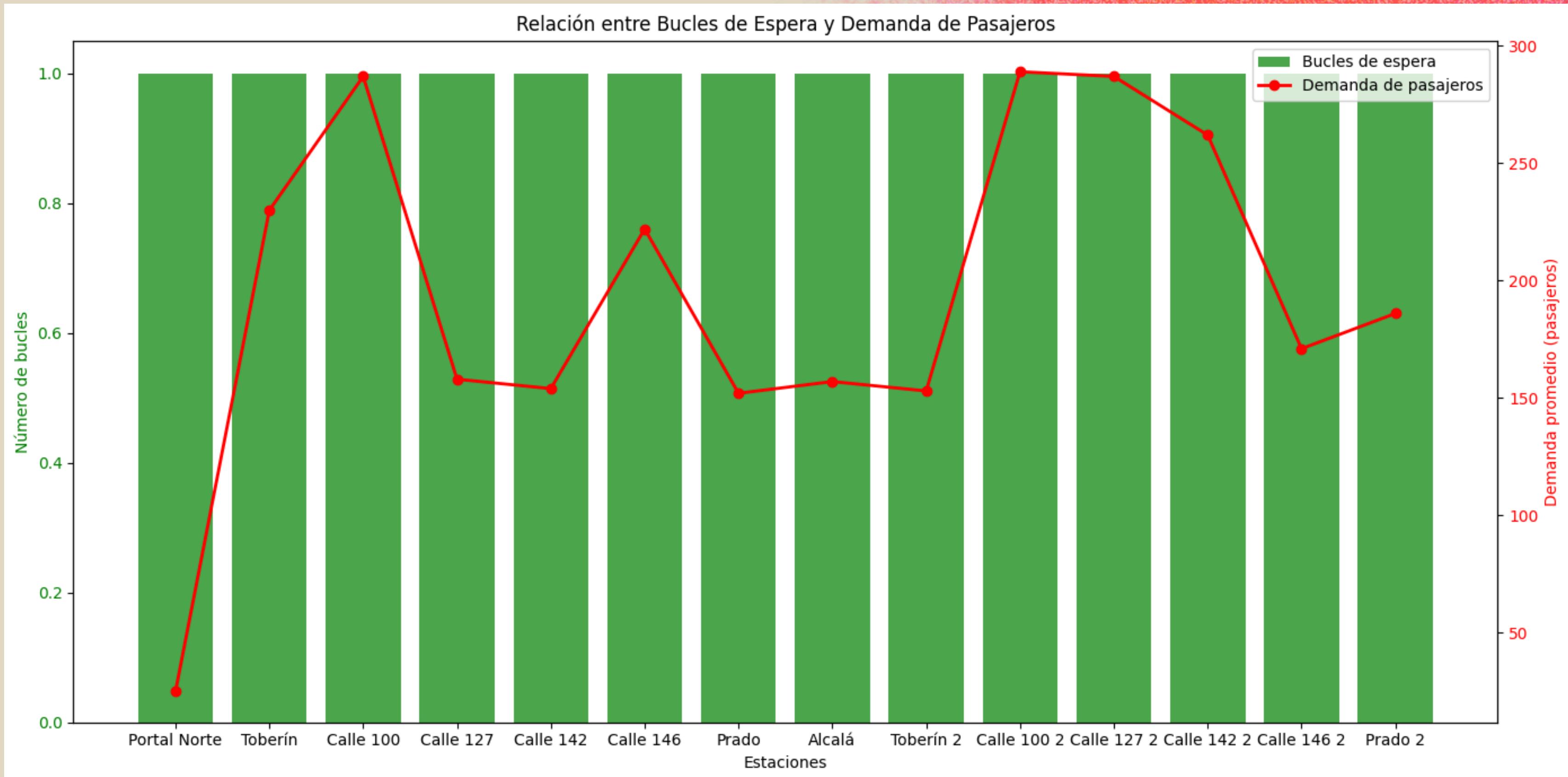
Tiempo total de recorrido: 4085.20 segundos (68.09 minutos)

Tiempo total de espera en estaciones: 295.30 segundos (4.92 minutos)

Eficiencia de la ruta (entropía/dimensión): 0.016997

Distribución de bucles de espera por estación: Portal Norte: 1 bucles (tiempo de espera: 48.20s) Toberín: 1 bucles (tiempo de espera: 19.80s) Calle 100: 1 bucles (tiempo de espera: 17.60s) Calle 127: 1 bucles (tiempo de espera: 11.20s) Calle 142: 1 bucles (tiempo de espera: 19.70s) Calle 146: 1 bucles (tiempo de espera: 20.30s) Prado: 1 bucles (tiempo de espera: 12.90s) Alcalá: 1 bucles (tiempo de espera: 21.80s) Toberín 2: 1 bucles (tiempo de espera: 20.70s) Calle 100 2: 1 bucles (tiempo de espera: 21.20s) Calle 127 2: 1 bucles (tiempo de espera: 21.30s) Calle 142 2: 1 bucles (tiempo de espera: 20.70s) Calle 146 2: 1 bucles (tiempo de espera: 16.50s) Prado 2: 1 bucles (tiempo de espera: 23.40s)

### Relación entre Bucles de Espera y Demanda de Pasajeros



# SUGERENCIAS DE OPTIMIZACIÓN

Estaciones críticas (alto tiempo de espera y demanda):

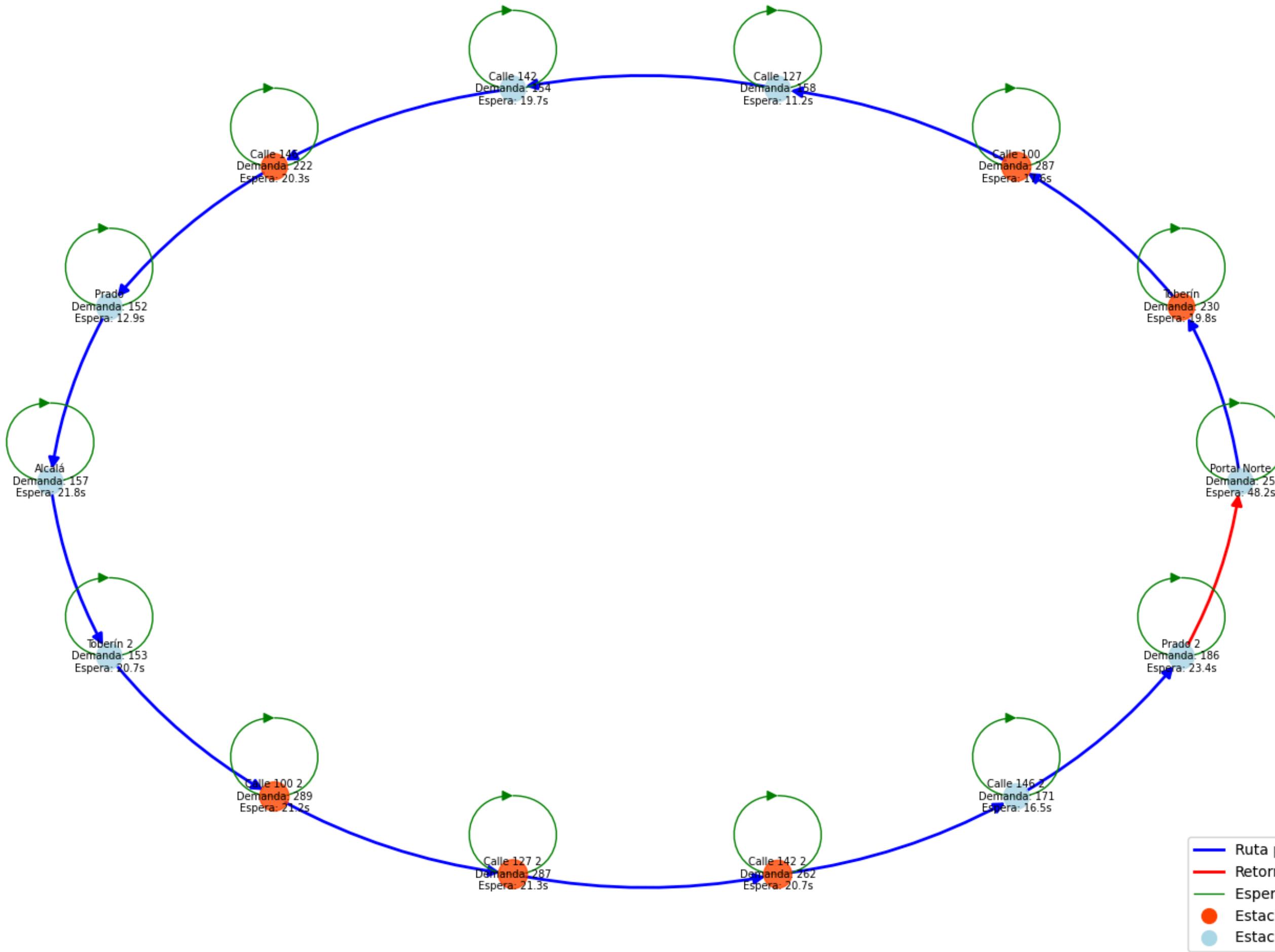
- Toberín: Búcles = 1, Tiempo de espera = 19.80s, Demanda = 230
- Calle 100: Búcles = 1, Tiempo de espera = 17.60s, Demanda = 287
- Calle 146: Búcles = 1, Tiempo de espera = 20.30s, Demanda = 222
- Calle 100 2: Búcles = 1, Tiempo de espera = 21.20s, Demanda = 289
- Calle 127 2: Búcles = 1, Tiempo de espera = 21.30s, Demanda = 287
- Calle 142 2: Búcles = 1, Tiempo de espera = 20.70s, Demanda = 262

## Sugerencias

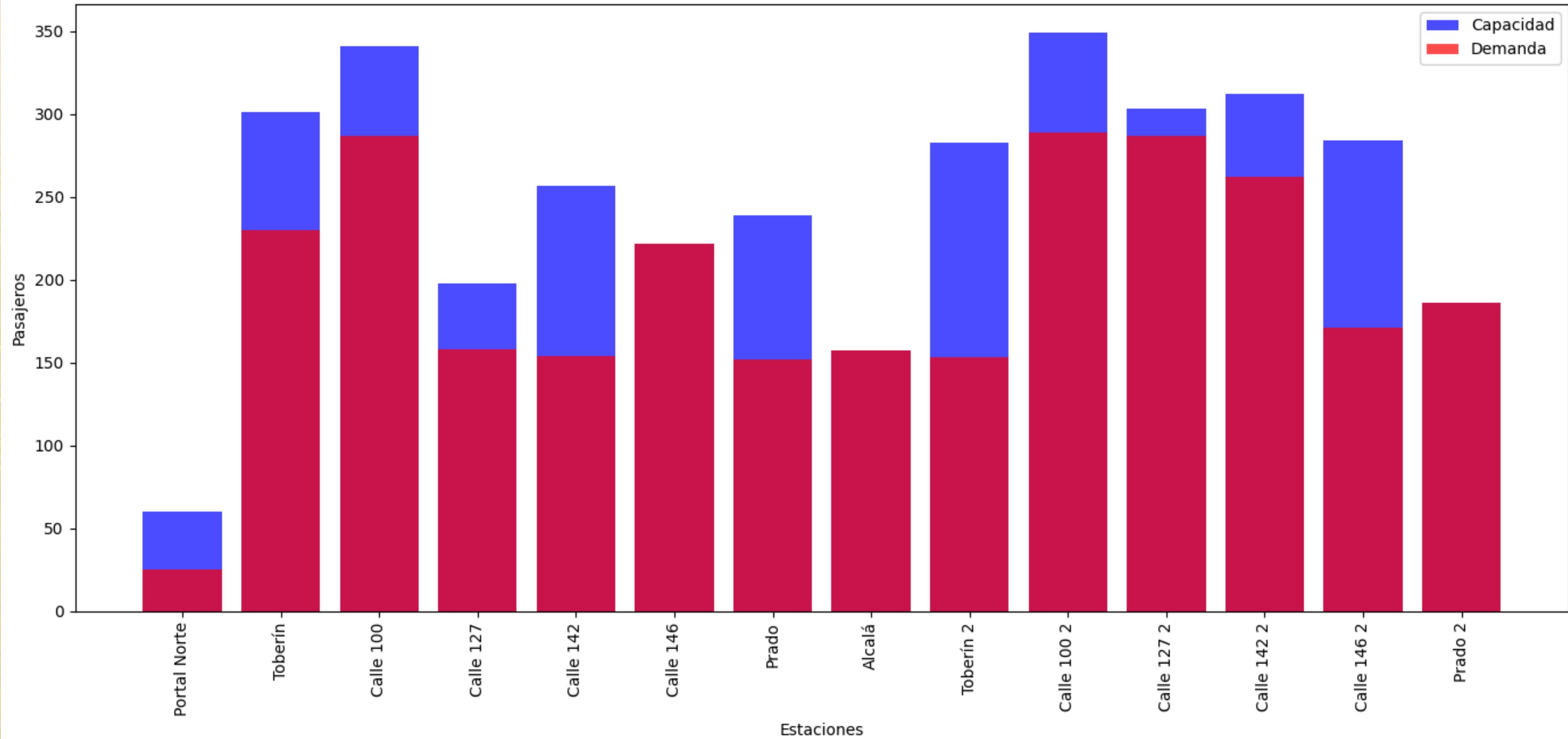
Posibles optimizaciones del ciclo:

1. Reducir tiempos de espera en estaciones críticas
2. Optimizar ruta de retorno (última a primera estación)

### Estructura Cíclica de la Ruta B12 con Bucles de Espera



Comparación entre Capacidad y Demanda por Estación



# MEJORAS Y DESAFIOS

- Hacer la construcción con la ruta completa.
- Mejorar el modelo de optimización tomando más variables.
- Aumentar recursos computacionales para mejorar la carga del código

# B I B L I O G R A F Í A

- E. A. Rodríguez and J. S. Suárez, Introducción a las álgebras de configuración de Brauer, 2021.  
Disponible en: <http://hdl.handle.net/11349/28649>.
- A. M. Cañas, G. B. Ríos, and I. D. M. Gaviria, Brauer Configuration Algebras Arising from Dyck Paths, Mathematics 10 (2022), no. 9, 1378. <https://doi.org/10.3390/math10091378>.
- E. L. Green and S. Schroll, Brauer configuration algebras: A generalization of Brauer graph algebras, Bull. Sci. Math. 141 (2017), no. 6, 539–572. <https://doi.org/10.1016/j.bulsci.2017.06.001>.

